A NOVEL APPROACH FOR OFF-LINE MULTIPROCESSOR SCHEDULING IN EMBEDDED HARD REAL-TIME SYSTEMS

Raimundo Barreto[1], Paulo Maciel[1], Marília Neves[1], Eduardo Tavares[1], Ricardo Lima[2]
[1] *Centro de Informática (CIn) at Universidade Federal de Pernambuco (UFPE). PO Box 7851, 50732-970 Recife-PE-Brazil.*
{ rsb,prmm,mln2,eagt } @cin.ufpe.br.
[2] *Departamento de Engenharia da Computação - Universidade de Pernambuco Recife-PE-Brazil*
ricardo@upe.poli.br

Abstract: There are two general approaches for scheduling tasks in real-time systems: runtime and pre-runtime scheduling. However, there are several situations where the runtime approach does not find a feasible schedule even if such a schedule exists. The proposed approach uses state space exploration for finding a pre-runtime scheduling. The main problem with such methods is the space size, which can grow exponentially. This paper shows how to minimize this problem, and presents a depth-first search method on a timed labeled transition system derived from the time Petri net model.

Keywords: Embedded real-time systems, scheduling, formal methods, and time Petri nets.

1. INTRODUCTION

Embedded hard real-time systems are dedicated computer applications having to satisfy stringent timing constraints. For meeting this requirement, scheduling performs an important role. There are two general approaches for scheduling tasks: runtime and pre-runtime scheduling. In *runtime scheduling,* the schedule is computed on-line as tasks arrive, using a priority-driven approach. However, there are situations where this approach may constrain the possibility of finding a feasible schedule, even if such schedule exists [10–11]. The approach presented in this paper is *pre-runtime scheduling,* where schedules are computed entirely off-line. This solution reduces context switching, its execution is predictable, and excludes the need of complex operating

systems. In safety-critical systems the predictability is an important matter, mainly due to the use of arbitrary precedence and exclusion relations. In accordance with [11], pre-runtime scheduling is often the only means of providing predictability in complex systems. This work uses *state space exploration* since it presents a complete automatic strategy for verifying finite-state systems [6]. In spite of the fact that a scheduling can be found using this strategy, this may be limited by the excessive size of its state space. The proposed approach tackles this problem by applying techniques for state space reduction, and a depth-first search algorithm. This paper is an extension of ours previous work [3], which presents how to reach feasible schedules by using a time Petri net model on uniprocessor architectures.

2. RELATED WORK

Xu and Parnas [10]present a branch-and-bound algorithm that finds an optimal pre-runtime schedule on a single processor for real-time process segments with release, deadlines, and arbitrary exclusion and precedence relations. Despite the importance of their work, it does not present real-world experimental results. Abdelzaher and Shin [1]proposed an extension of [10]in order to deal with distributed real-time systems. This algorithm takes into account delays, precedence relations imposed by interprocess communications, and considers many possibilities for improving the scheduling lateness at the cost of complexity. The scheduler synthesis proposed by Altisen et.al. [2]synthesizes all *dynamic on-line scheduling* satisfying a given property. In spite of they have claimed that using *synchronization modes* the complexity is reduced, they do not directly address the state explosion problem, stressed by the authors as a limitation of their approach. Several authors also use Petri nets in scheduling theory. However, most of them are only concerned with schedulability analysis. For instance, Bruno et. al. [4]present a schedulability analysis, using high-level Petri nets. However, their work does not generate feasible schedules, but it relies on Xu and Parnas' algorithm [10]in order to find them.

Comparing our approach with other works (e.g., [1, 9]), it differs in the sense that: (i) their works model the *scheduling problem,* whilst our work models the tasks of a system. For this reason, they may have better performance in some situations. Nevertheless, time efficiency is not a critical concern when considering schedules computed off-line. However, our solution can also generate timely and predictable scheduled code, which is difficult in their works. (ii) using Petri net analysis techniques allows one to check several system properties. Although state space exploration is not new, at the best of our present knowledge, there is no similar work that uses formal methods for modeling real-time systems, and finds a feasible pre-runtime schedule considering multiprocessor architectures.

3. COMPUTATIONAL MODEL: SYNTAX AND SEMANTICS

The computational model syntax is given by a time Petri net [7], which is a Petri net extended with time, and its semantics is given by its time labeled transition system. A time Petri net (TPN) is a bipartite directed graph represented by a tuple $\mathcal{P}= (P, T, F, W, m_0, I)$. P (places), and T (transitions) are two types of nodes. The edges are represented by $F \subseteq (P \times T) \cup (T \times P)$. $W : F \rightarrow \mathbb{N}$ represents the weight of the edges. A TPN marking m_i is a vector $m_i \in \mathbb{N}^{|P|}$, and m_0 is the initial marking. $I : T \rightarrow \mathbb{N} \times \mathbb{N}$, represents the timing constraints, where $I(t) = (EFT(t), LFT(t)) \ \forall t \in T$. $ET(m_i)$ is a set of enabled transitions in marking m_i. Let M be the set of all reachable markings of $\mathcal{P}$. $C \in \mathbb{N}^{|ET(M)|}$ is a clock vector, which represents the time elapsed since the respective transition enabling. In order to facilitate the TPN's analysis, it is defined the dynamic firing interval $(I_D(t) = (DLB, DUB)$, where $DLB(t) = max(0, EFT(t) - c(t))$ and $DUB(t) = LFT(t) - c(t)$. $I_D(t)$ is dynamically modified whenever the respective clock variable is incremented, and t does not fire.

The set of states S of $\mathcal{P}$ is given by $S \subseteq (M \times \mathbb{N}^{|ET(M)|})$, that is, a single state is defined by a pair (m, c), where m is a marking, and c is its respective clock vector for $ET(m)$. The initial state is $s_0 = (m_0, c_0)$, where $c_0(t) = 0 \ \forall t \in ET(m_0)$. $FT(s)$ is the set of firable transitions at state s defined by: $FT(s) = \{t_i \in ET(m) | DLB(t_i) \leq min(DUB(t_k)) \forall t_k \in ET(m)\}$, where $FT \subseteq ET \subseteq T$. The *firing domain* for t at a specific state s, is defined by: $FD_s(t) = [DLB(t), \min(DUB(t_k))], \forall t_k \in ET(m)$.

The semantics of a TPN $\mathcal{P}$ is defined by associating a TLTS $\mathcal{L}_\mathcal{P}= (S, \Sigma, \rightarrow, s_0)$ such that: (i) S is a finite set of discrete states of $\mathcal{P}$; (ii) $\Sigma \subseteq (T \times \mathbb{N})$ is an alphabet of labels representing activities. The labels are (t, θ) corresponding to the firing of a firable transition (t) at a specific time value (θ) in the firing interval $FD(s), \forall s \in S$; (iii) $\rightarrow \subseteq S \times \Sigma \times S$ is the transition relation; and (iv) s_0 is the initial state of $\mathcal{P}$.

Let $\mathcal{L}_\mathcal{P}$ be a TLTS derived from a time Petri net $\mathcal{P}$, and $s_i = (m_i, c_i)$ a reachable state. $s_j = \texttt{fire}(s_i, (t, \theta))$ denotes that firing a transition t at time θ from the state s_i, a new state $s_j = (m_j, c_j)$ is reached, such that:
(i) $\forall p \in P,\ m_j(p) = m_i(p) - W(p,t) + W(t,p)$;

$$\text{(ii) } \forall t_k \in ET(m_j), C_j(t_k) = \begin{cases} 0, & if(t_k = t) \\ 0, & if(t_k \in ET(m_j) - ET(m_i)) \\ C_i(t_k) + \theta, & else \end{cases}$$

The firing of a transition t_i, at a specific time θ_i in the state (s_{i-1}) defines the next state (s_i).

Let $\mathcal{L}_\mathcal{P}$ be a TLTS of a TPN $\mathcal{P}$, where s_0 its initial state, $s_n = (m_n, c_n)$ a final state, and $m_n = M^F$, which is the desired final marking. $s_0 \xrightarrow{(t_1,\theta_1)}$

$s_1 \xrightarrow{(t_2,\theta_2)} s_2 - - \rightarrow s_{n-1} \xrightarrow{(t_n,\theta_n)} s_n$ is defined as a *feasible firing schedule,* where $s_i = \texttt{fire}(s_{i-1}, (t_i, \theta_i)), i > 0$, if $t_i \in FT(s_{i-1})$, and $\theta_i \in FD_{s_{i-1}}(t_i)$. As it is presented later, the modeling methodology guarantees the final marking M^F is well-known since it is explicitly modeled.

4. TEST MODEL

Let $\mathcal{T}$ be the set of tasks in a system. Let τ_i be a periodic task defined by $\tau_i = (ph_i, r_i, c_i, d_i, p_i)$, where ph_i is the initial phase (delay associated to the first time request of a task after the system starting); r_i is the release time (interval between the beginning of a period and the earliest time that a task execution can be started); c_i is the worst case computation time; d_i is the deadline (interval between the beginning of a period and the time when the task must be completed); and p_i is the period (time interval in which the task must be executed). Let $\tau_k = (c_k, d_k, min_k)$ be a sporadic task, where c_k is the worst case computation time; d_k is the deadline; and min_k is the minimum period between two activations of task τ_k. A task is classified as sporadic if it can be randomly activated, but the minimum period between two activations is known. As pre-runtime approaches may only schedule periodic tasks, the sporadic tasks have to be translated to an equivalent periodic task [8]. A task τ_i *precedes* task τ_j, if t_j can only start execution after t_i has finished. A task τ_i *excludes* task τ_j, if no execution of t_j cannot start while task t_i is executing. Exclusion relations may prevent simultaneous access to shared resources. Each task $\tau_i \in \mathcal{T}$ consists of a finite sequence of *task time units* $\tau_i^0, \tau_i^1, \cdots, \tau_i^{c_i-1}$, where τ_i^{j-1} always precedes τ_i^j, for $j > 0$. A task time unit is the smallest indivisible granule of a task, during which it cannot be preempted by any other task. A task can also be split into more than one *subtasks,* where each subtask is composed by one or more task time units.

5. MODELING REAL-TIME SYSTEMS

Hard real-time systems are those that besides its functional correctness, timeliness must be satisfied. The modeling phase is very important to attain such constraints.

5.1 Scheduling Period

The proposed method schedules the set of periodic tasks occurring in a period that is equal to the least common multiple (LCM) of the periods of the given set of tasks. The LCM is also called *schedule period* (P_S). Within this new period, there are several *tasks instances* of the same task, where $N(t_i) = P_S/p_i$ gives the instances of t_i. For example, consider the following task model consisting of two tasks: $t_1 = (0, 0, 2, 7, 8)$ and $t_2 = (0, 2, 3, 6, 6)$.

In this particular case, $P_S = 24$, implying that the two periodic tasks are replaced by seven new periodic tasks ($N(t_1) = 3$, and $N(t_2) = 4$), where the timing constraints of each task instance has to be transformed to consider that new period [10].

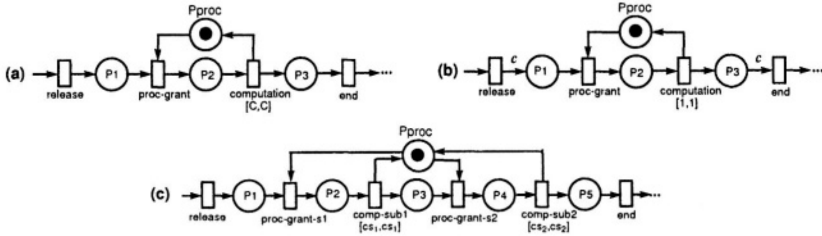

Figure 1. Modeling Scheduling Methods

5.2 Scheduling Methods

Figure 1 presents three ways for modeling scheduling methods, where $c = cs_1 + cs_2$ is the task computation time (cs_1 and cs_2 are computation times for the first and last subtask, respectively):

a) *all-non-preemptive:* processor is just released after the entire computation be finished. Figure 1(a) shows that computation transition timing interval has bounds equal to the task computation time (i.e., [*c*, *c*]);

b) *all-preemptive:* tasks are implicitly split into all possible subtasks. This method allows running other *conflicting tasks,* meaning that one task could preempt another task. It is worth observing, the difference between the timing interval for the computation transition and the arc weight in Figures 1(a) and 1(b).

c) *defined subtasks:* tasks are split into more than one explicitly defined subtasks. Figure 1(c) shows two subtasks.

5.3 Tasks Modeling

Figure 2 is also used to show (in dashed boxes) the three main *building blocks* for modeling a real-time task. These blocks are: *(a) Task Arrival,* which models the periodic invocation for all task's instances. Transition t_{ph} models the initial phase, whilst transition t_a models the periodic arrival for the remaining instances; *(b) Deadline Checking,* where it is used elementary net structures to capture deadline missing. Some works (e.g. [2]) extended the Petri net model for dealing with deadline checking. *(c) Task Structure,* which

models: release time, processor granting, computation, and processor releasing. Figure 2 presents a non-preemptive TPN model for the example presented in previous subsection. It does not model the seven task instances. Instead, it models only the two original tasks, and the time period of every task instances.

5.4 Modeling Interprocessor Communication

Processors are connected to one (or more) bus, which is modeled as a resource that is shared by all processors and accessed in mutual exclusion. The proposed approach schedules the communication for avoiding network contention. Otherwise, it could result in different execution times for different runs of the same system, which is not appropriated for hard real-time systems. It is supposed that: (i) the communication time between tasks in the same processor is negligible; and (ii) the communication is synchronous (blocking). Figure 3 presents a model for two interprocessor communicating tasks (`ping` and `pong`). The task `ping` computes and sends a data to `pong`. When the data arrives, the task `pong` computes and sends a new data to `ping`, and this

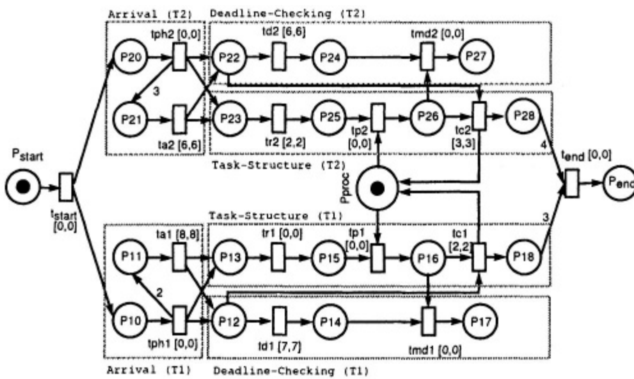

Figure 2. Petri net model

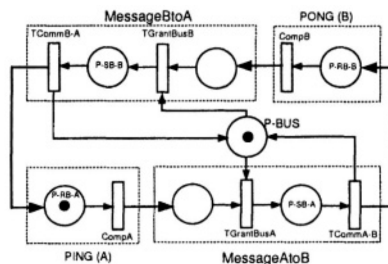

Figure 3. A Simple Example of Interprocessor Communication

```
1  scheduling-synthesis(S,M^F,TPN)
2  {
3    if (S.M = M^F) return TRUE;
4    tag(S);
5    PT = remove-undesirable(partial-order(firable(S)));
6    if (|PT| = 0) return FALSE;
7    for each (⟨t,θ⟩ ∈ PT) {
8      S'= fire(S, t, θ);
9      if (untagged(S') ∧ scheduling-synthesis (S',M^F,TPN)){
10       add-in-trans-system (S,S',t,θ);
11       return TRUE;
12     }
13   }
14   return FALSE;
15 }
```

Figure 4. Scheduling Synthesis Algorithm

procedure repeats indefinitely. The bus is modeled by a place (P−BUS) shared by all tasks. The communication time is attached to transitions TCommA−B and TCommB−A. The places P−SB−A and P−SB−B model sending buffers, whilst places P−RB−A and P−RB−B model receiving buffers.

6. PRE-RUNTIME SCHEDULING

This section shows a technique for state space minimization, the algorithm that implements the proposed method, and an application of the algorithm.

6.1 Minimizing State Space Size

Partial-Order Reduction. If activities can be executed in any order, such that the system always reaches the same state, these activities are *independent.* Partial-order reduction methods exploit the independence of activities [6]. An independent activity is one that is not in conflict with other activity, that is, when it is executed it does not disable any other activity, such as: arrival, release, precedence, computation, processor releasing, and so on. This reduction method proposes to give for each class of activities a different *choice-priority.* Dependent activities, like processor granting and exclusion relations, have lowest priority. Therefore, when changing from one state to another, it is sufficient to analyze the class with highest choice-priority and pruning the other ones. This reduction is important due to two reasons: (i) it reduces the amount of storage; and (ii) when the system does not have a feasible schedule, it returns more rapidly.

Undesirable States. Section 5 presents how to model undesirable states, for instance, states that represent missed deadlines. The proposed method is interested for schedules that do not reach any of these undesirable states.

6.2 Pre-Runtime Scheduling Algorithm

The algorithm proposed (Fig. 4) is a depth-first search method on a TLTS. The *stop criterion* is obtained whenever the desirable final marking M^F is reached. Considering that, (i) the Petri net model is guaranteed to be bounded, and (ii) the timing constraints are bounded and discrete, this implies that the TLTS is finite and thus the proposed algorithm always finishes. When the algorithm reaches the desired final marking (M^F), it implies that a feasible schedule was found (line 3). The state space generation is modified (line 5) to incorporate the state space pruning. `PT` is a set of ordered pairs $\langle t, \theta \rangle$ representing for each firable transition (post-pruning) all possible firing time in the firing domain. The *tagging scheme* (lines 4 and 9) ensures that no state is visited more than once. The function `fire` (line 8) returns a new generated state (S') due to the firing of transition t at time θ. The feasible schedule is represented by a TLTS generated by the function `add-in-trans-system` (line 10). The whole reduced state space is visited only when the system does not have a feasible schedule.

Table 1. Illustrative example

st	PT	trans+time	st	PT	trans+time	st	PT	trans+time
0	{tstart}	{tstart,0}	12	{tr2}	{tr2,0}	19	{tp2}	{tp2,0}
1	{tph1,tph2}	{tph1,0}	**13**	**{tp1,tp2}**	**{tp1,0}**	20	{ta1}	{ta1,2}
2	{tph2}	{tph2,0}	14	{tc1}	{tc1,2}	21	{tr1}	{tr1,0}
3	{tr1}	{tr1,0}	15	{tp2}	{tp2,0}	22	{tc2}	{tc2,1}
4	{tp1}	{tp1,0}	16	{ta2}	{ta2,2}	23	{tp1}	{tp1,0}
5	{tr2}	{tr2,2}	**17**	**{td2}**	**{td2,0}**	24	{ta2}	{ta2,1}
6	{tc1}	{tc1,0}	**13**	**{tp2}**	**{tp2,0}**	25	{tc1}	{tc1,1}
7	{tp2}	{tp2,0}	14	{tc2}	{tc2,3}	26	{tr2}	{tr2,1}
8	{tc2}	{tc2,3}	15	{tp1}	{tp1,0}	27	{tp2}	{tp2,0}
9	{ta2}	{ta2,1}	16	{ta2}	{ta2,1}	28	{tc2}	{tc2,3}
10	{ta1}	{ta1,2}	17	{tc1}	{tc1,1}	**29**	**{tend}**	**{tend,0}**
11	{tr1,tr2}	{tr1,0}	18	{tr2}	{tr2,1}			

6.3 Application of the Algorithm

Table 1 depicts the execution of the algorithm applied to the time Petri net model of Figure 2. In this table, at state 13, two transitions (tp_1 and tp_2) are firable. The possible execution of task $T1$ (choosing tp_1 for firing) is a wrong choice since, after that, task $T2$ misses its deadline (state 17). The algorithm *backtracks* to state 13 and try the alternative, now granting the processor to the task $T2$ (firing tp_2). This new decision leads to a feasible schedule, since in the state 29 the firing of transition t_{end} reaches the desired final marking (M^F).

Table 2. **Experimental results summary**

Example	instances	state-min	found	time (s)
Simple Control Appl	**28**	**50**	**50**	**0.005**
Robotic Arm	37	150	150	0.03
Xu (example 3)	4	171	1566	0.82
Xu (figure 9)	5	281	2387	3.41
Mine Pump Control	782	3130	3255	11.6

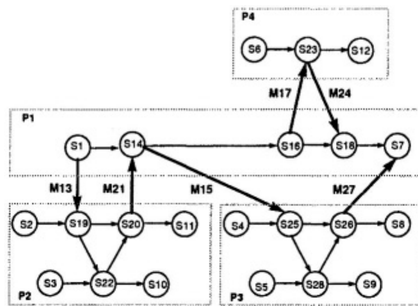

Figure 5. Case Study Graph

7. EXPERIMENTAL RESULTS

Table 2 shows a summary of the experimental results. All experiments were performed on a Pentium-III 600 Mhz dual processor. In order to depict the practical usability of the proposed method in more details, one of the examples is considered, *a simple control application.* This case study is described originally in [5]. The system consists of a sensory device mounted on a motorized platform that must detect and track specific objects in the environment. Four processors connected by a single bus control the system. The model consists of 6 tasks split into 22 subtasks, which exchanges 10 messages, 6 of them are sent across processor boundaries. Figure 5 shows the computational graph for this application, presenting the subtasks allocated to processors, and its communication pattern. In this graph each node is labeled with the corresponding subtask number, arcs representing local communication are treated as precedence relation, and each arc representing an interprocessor communication is labeled with a corresponding *message identification.* The proposed algorithm founds a feasible scheduling with no overhead, since it only examined the minimum number of states (in this case 50 states) in 5 milliseconds.

8. CONCLUSIONS

This paper proposed a formal modeling methodology based on time Petri nets, and a framework for pre-runtime scheduling on multiprocessors using a reduced state space exploration algorithm. In spite of this analysis technique is not new, to the best of our knowledge, there is no work reported similar to ours that models hard real-time systems and finds (whether one exists) a respective pre-runtime scheduling. The real-time task specification can be very general, since it can have resource and timing constraints, and intertask relations, such as precedence and exclusion relations. The proposed algorithm is a depth-first search method on a finite TLTS derived from a TPN model. When searching for a feasible schedule, the algorithm suffers from the state space explosion problem. In order to maintain the state space growth under control, the proposed method uses minimization techniques. The algorithm presented here always finds a schedule, provided that one exists.

The proposed modeling and the scheduling synthesis are an important step toward embedded real-time software synthesis tools. So, it is planned to generate complete executable code from the formal model. This can be solved through TPN with tasks, which is an extension of TPN, which annotates transitions with program code. Another extension is to take into account different operational modes in the pre-runtime scheduling.

REFERENCES

[1] T. F. Abdelzaher and K. G. Shin. Optimal combined task and message scheduling in distributed real-time systems. In *Proc. IEEE RTSS,* pages 162–171, December 1995.

[2] K. Altisen, G. Göbler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. *IEEE Real-Time System Symposium,* pages 154–163, Dec 1999.

[3] R. Barreto, S. Cavalcante, and P. Maciel. A time petri net approach for finding pre-runtime schedules in embedded real-time systems. In *1st Int. Workshop on Embedded Computing Systems (ECS'04).* IEEE CS Press, march 2004.

[4] G. Bruno, A. Castella, G. Macario, and M. Pescarmona. Scheduling hard real time systems using high-level petri nets. *13th ICATPN,* pages 93–112, Jun 1992.

[5] M. DiNatale and J. A. Stankovic. Dynamic end-to-end guarantees in distributed realtime systems. In *Proc. of the IEEE Real-Time Systems Symposium,* pages 216–227, 1994.

[6] P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem.* PhD Thesis, University of Liege, Nov. 1994.

[7] P. Merlin and D. J. Faber. Recoverability of communication protocols: Implicatons of a theoretical study. *IEEE Transactions on Communications,* 24(9):1036–1043, Sept. 1976.

[8] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment.* PhD Thesis, MIT, May 1983.

[9] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Soft. Engineering,* 19(2):139–154, February 1993.

[10] J. Xu and D. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Soft. Engineering,* 16(3):360–369, March 1990.

[11] J. Xu and D. Parnas. On satisfying timing constraints in hard real-time systems. *IEEE Trans. Soft. Engineering,* 1(19):70–84, January 1993.